Oto : Un outil générique et évolutif d'aide à la correction de programmes -- Langage de scriptage

Note interne de recherche

Département d'informatique, UQAM

F. Guérin et G. Tremblay

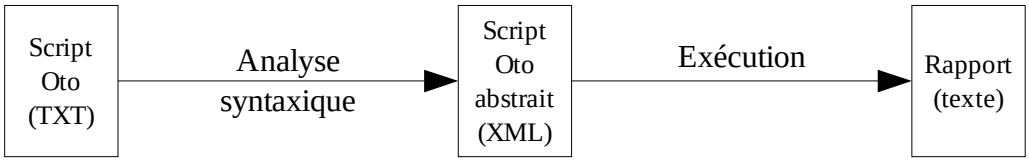
Juin 2004

Table des matières

1 Le système de correction Oto

Oto est une application informatique qui permet d'automatiser en partie la correction des travaux des étudiants. Cette correction peut se faire « en batch », par le correcteur, ou de facon immédiate et en ligne, par l'étudiant (correction préliminaire partielle). Les modalités d'une correction sont spécifiées dans un programme dit script Oto (appelée aussi une « évaluation »).

La figure suivante détaille les étapes du processus d'exécution d'un script Oto. Toutefois, ces détails sont invisibles pour l'utilisateur normal et ne sont présentés que pour tenter d'éclairer les différents niveaux de la mise en oeuvre d'Oto ainsi que certaines de ses caractéristiques (par exemple, l'évaluation d'expressions complexes).



Le script est d'abord analysé syntaxiquement pour produire une représentation abstraite d'un script exécutable. Ce script est ensuite exécuté. Un rapport d'exécution est finalement produit.

2 Structure générale d'un script Oto

Un script Oto se compose de :

- Tâches à accomplir
- Données internes (variables) ou externes (fichiers)

2.1 Tâches Oto

Un script Oto se compose principalement d'une série de tâches à accomplir. Chaque tâche a un nom, un composant qui est utilisé pour accomplir la tâche, des paramètres utilisés par ce composant pour accomplir ladite tâche ainsi qu'un résultat dénoté par le nom de la tâche.

Prenons par exemple le cas suivant, décrit à l'aide d'une syntaxe relativement abstraite (une syntaxe concrète plus concise sera présentée dans les exemples):

```
tache {
   nom = compilation
   composant = java/compile
   parametres = {
      mainFile = tp3.java
   }
}
```

Ici, la tâche à accomplir est la «compilation», à l'aide du composant «java/compile», en utilisant un paramètre nommé «mainfile» qui indique typiquement le fichier racine du projet Java.

2.1.1 Utiliser le résultat fourni par une tâche précédente

Le résultat de l'exécution d'une tâche est automatiquement conservé, par l'intermédiaire d'une varible, pour utilisation ultérieure dans le script. Cette variable porte le même nom que celui donné à la tâche. Dans l'exemple suivant on effectue des tests sur le résultat de la compilation.

```
tache {
   nom = analyse
   composant = java/test
   parametres = {
      testedFile = $compilation.mainClass
      junitFile = tp3test.class
   }
}
```

Dans ce cas-ci, le composant `java/test` est invoqué. Lors de la compilation (voir section précédente), le composant `java/compile` a produit un résultat intermédiaire stocké dans la variable `$compilation`. En fait, il s'agit plutôt d'une collection de résultats, chacun accessible par l'intermédiaire d'un identificateur. L'un d'entre eux a pour nom `mainClass` et désigne le nom de la classe principale qui doit être invoquée pour exécuter le programme Java une fois compilé.

2.2 Expressions simples

Dans la description d'une tâche, chaque paramètre du composant est associé à une valeur. Cette valeur est le fruit d'une expression. Une expression peut être :

- Une valeur littérale interprétée comme une simple chaîne, e.g., `tp3.class`.
- Une référence à une variable, e.g., `$compilation.mainClass`.

  Note : La portée de la déréférence peut être indiquée avec «`{...}`», e.g., `${oto.repScript}/tests/testbase.class`.
- Une chaîne de caractères dans laquelle des substitutions de variables seront effectuées, e.g., `"java $compilation.mainClass"`.

2.3 Expressions complexes

Il est possible d'évaluer des expressions plus complexes en utilisant l'interpréteur sur lequel est basé Oto (plus précisément, Ruby). Le résultat est toujours une chaîne de caractère (conversion implicite utilisant "to_s"). L'expression suivante calcule la valeur de l'expression arithmétique « 2 + 2 ».

```
$( 2 + 2 )
```

2.4 Variable

La valeur d'une expression peut aussi être calculée et stockée dans une variable pour être référencée ultérieurement. Notez bien qu'il s'agit ici de « variables » à affectation unique (style langage fonctionnel).

```
pi = 3.1416
_2pi = $( 2 * $pi )
```

2.5 Assertion

Une assertion peut être évaluée pour vérifier l'état du script. Si la condition associée à l'assertion est fausse, alors le script échoue et la cause de l'erreur est rapportée.

```
assurer {
   condition = $( FileTest.exist $tp3 )
   msg = "Le fichier $tp3 est introuvable."
}
```

L'assertion échoue si la valeur de l'expression est "false". Sinon elle réussit.

2.6 Variable globale $oto

Un script Oto s'exécute dans un certain contexte. Les informations relatives à ce contexte sont stockées dans la variable multivaluée $oto, qui est automatiquement initialisée avant le début de l'exécution proprement dite du script. Les attributs de cette variable sont (liste incomplète et temporaire *****) :

Attribut	***Description***	***Exemple***
repTravail	Répertoire de travail alloué pour l'exécution du script et accessible en lecture et écriture. Cet espace de travail est initialisé au préalable avec le contenu du TP remis par l'équipe étudiante et avec l'échafaudage prévu pour cette évaluation.	`/home/otoetudiant1/travail`

Attribut	*Description*	*Exemple*
equipe	Nom de l'équipe à qui appartient le TP. Ce nom se compose des identificants des membres de l'équipe, en ordre alphanumérique et séparés par des « : ».	`KHAG12345678:BONN87654321`
repTps	Répertoire qui contient tout les TPs de chacune des équipes. Accessible en lecture seulement. Chaque sous-répertoire porte le nom d'une équipe et contient le travail de cette équipe. Ce répertoire est accessible en lecture seulement.	`/home/otoetudiant1/tps`
repScript	Répertoire qui contient le script et les fichiers attachés au script (par exemple, des jeux de tests ou des fichiers de tests). Accessible en lecture seulement.	`/home/otoetudiant1/script`
dateRemise	Date de remise du TP en format `aa-mm-jj@hh:mm`. Peut servir à appliquer une pénalité de retard.	`04-03-28@17:00`
evaluation	Nom de l'évaluation.	`tp3`
groupe	Nom du groupe.	`INF1105-10-hiv04`
uidEnseignant	Nom d'usager de l'enseignant.	`cesar`
nomEnseignant	Nom de l'enseignant.	`Jules Cesar`

2.7 Variable de type fichier

Un cas fort répandu dans un script Oto est de spécifier un identificateur dénotant un nom de fichier, fichier qui doit nécessairement exister pour que le script puisse fonctionner. Oto inclut donc un mécanisme spécifique pour ce faire.

```
tp3 =? tp3.java
```

La partie droite de l'opérateur « `=?` » est une expression simple ou complexe. L'expression est évaluée et le résultat est alors considéré comme un nom de fichier. Si ce nom de fichier est spécifié par un chemin relatif et non absolu, le chemin est complété pour le rendre absolu. Si le fichier dénoté par ce nom existe, alors le nom du fichier est associé à l'identificateur (partie gauche de « `=?` »). Si le fichier n'existe pas, alors le script échoue. Les chemins relatifs sont toujours supposés donnés par rapport au répertoire de travail affecté au script par Oto.

2.8 Unicité des noms

Les tâches et variables doivent avoir des noms différents.

2.9 Sortie du script

La sortie est spécifiée par une instruction conséquente:

```
sortir { variable1, variable2, variable3 }
```

N.B. Les résultats des tâches ne sont pas directement sortables mais doivent être stockés dans une variable au préalable.

2.10 Exemple complet

Cet exemple fait intervenir les notions vues précédemment. Remarquez la notation plus compacte (syntaxe concrète). Le symbole « # » dénote le début d'un commentaire, qui se termine à la fin de la ligne.

```
# L'évaluation se fait sur 100 pts
total_sur = 100

# Le fichier source fourni par l'équipe
fmain =? tp3.java

# Le fichier test fourni par l'enseignant
ftest =? $oto.repscript/tests/testbase.class

# On compile le TP de l'étudiant
compilation :: java/compile {
   mainfile = $fmain
}

# On s'assure que la compilation a reussi
assurer $( $compilation.nb_erreurs == 0 )
sinon "Le TP ne compile pas et l'evaluation ne peut se poursuivre."

# Puis on utilise JUnit pour tester le programme de l'etudiant
analyse :: java/test {
   fcible =   $compilation.mainclass
   ftest =    $ftest
}

# On calcule la note en fonction du nombre d'erreurs détectées
note = $( $total_sur – $total_sur * $analyse.nb_erreurs /
$analyse.maxerrors )
```

```
# On regarde pour détecter les cas possibles de plagiat.
plagiat :: text/comparer {
   equipe = $oto.equipe
   tps = $oto.tps
   fichiers = $fmain
}

# On retient les cas suspects
cas_suspects = $plagiat.cas_suspects

sortir { note, cas_suspects }
```

3 Le rapport

Typiquement, l'exécution d'un script se termine par la production d'un rapport d'exécution. On retrouve deux types de rapport. Un rapport de groupe et un rapport d'équipe.

3.1 Rapport d'équipe

Ce rapport comprend les éléments suivants:

- En-tête (voir l'exemple plus loin).
- Résultat de l'exécution (ou message d'erreur).
- Sommaire de l'exécution.
- Détails de l'exécution.

3.1.1 Résultat de l'exécution

Le résultat de l'exécution est entièrement déterminé par la clause de sortie du script. Chacune des variable spécifiées dans cette clause apparaît comme un résultat du script. Si le script n'a pas de sortie, alors aucune sortie n'est produite. Si le script ne peut s'exécuter correctement, alors un message d'erreur apparaît.

3.1.2 Sommaire de l'exécution

Le sommaire de l'exécution du script indique la valeur produite par chaque tâche et variable du script, à la condition que cette tâche ou variable soit déclarée publique. Si elle est privée, elle n'apparaît pas dans le sommaire. Typiquement, la partie sommaire permet à l'enseignant ou à l'étudiant qui consulte le rapport de voir d'un seul coup d'oeil quelles tâches ont mis à jour certaines lacunes dans le travail. Les variables publiques, quant à elles, permettent de mettre en évidence certains paramètres importants pris en compte lors de l'évaluation, comme par exemple les poids affectés aux différentes parties de l'évaluation lors du calcul de la note.

Afin de rendre ce sommaire plus convivial, toute variable et tâche peut être accompagnée d'une description qui apparaîtra dans le rapport. Cette description peut aussi être utile pour les variables et tâches privées (voir plus loin pourquoi).

Une tâche ou variable publique apparaît avec un plus (« + ») en avant de son nom. Un moins (« - ») indique une tâche ou variable de caractère privée. Le signe « - » est facultatif puisque par défaut tout est privé.

3.1.3 Détails de l'exécution

Chaque tâche, en plus de produire un ensemble de valeurs, produit un rapport de tâche. Ce rapport apparaît dans la section DÉTAILS D'EXÉCUTION pour chaque tâche marquée publique, par exemple, la sortie d'une tâche de compilation comprendra probablement la liste des erreurs.

3.1.4 Exemple d'un script

Le script suivant est le même que le précédent, mais cette fois il a été décoré pour prendre en compte la production d'un rapport. Une description précède donc chaque tâche et variable. Un signe plus ou moins précédant le nom indique le caractère privé ou public du résultat ainsi que du commentaire de description associé.

```
# Quelques identificateurs.
<<Nombre total de points>>
   +total_sur = 100

<<Le fichier source fourni par l'equipe>>
   -fmain =? tp3.java

<<Le fichier test fourni par l'enseignant>>
   -ftest =? $oto.repscript/tests/testbase.class

# Les tâches.
<<Compilation du TP>>
   +compilation :: java/compile {
      mainfile = $fmain
   }

# On s'assure que la compilation a reussi
   assurer $( $compilation.nb_erreurs == 0 )
   sinon "Le TP ne compile pas et l'evaluation ne peut se poursuivre."

<<Tests JUnit du TP>>
   +analyse :: java/test {
      fcible = $compilation.mainclass
      ftest =  $ftest
   }

<<Note sur $total_sur>>
   # La note est calculée en fonction du nombre d'erreurs.
   +note = $( $total_sur – $total_sur *
      $analyse.nb_erreurs / $analyse.nb_tests_total )
```

```
<<Détection de plagiat>>
   +plagiat = text/comparer {
      equipe = $oto.equipe
      tps = $oto.tps
      fichiers = $fmain
   }

<<Nombre de cas suspects de plagiat>>
   -nb_cas_suspects = $plagiat.nb_cas_suspects

sortir { note, nb_cas_suspects }
```

3.1.5 Exemple de rapport

Le script précédent produirait typiquement le rapport suivant:

```
*****************************************************
OTO : RAPPORT DE CORRECTION
*****************************************************

Professeur:   Jules Cesar (cesar)
Groupe-cours: inf1105-10-hiv04
Evaluation:   tp3

Equipe:        KAHG12345678:BONN87654321
Membres:       Gengis Khan (KAHG12345678)
               Napoleon Bonaparte(BONN87654321)

Fichier:      tp3.jar
Remis le:     2004-03-28
Taille:       12345 octets
Md5:          E3RT78...

Script:        tp3_correction.oto
Exécuté le:   2004-04-01 a 13h34
Sur:          arabica
Par:          cesar (Jules Cesar)


-----------------------------------------------------
RESULTATS
-----------------------------------------------------

Total sur 100: 80

Nombre de cas suspects de plagiat: 1

-----------------------------------------------------
SOMMAIRE D'EXECUTION
-----------------------------------------------------
```

```
Nombre total de points: 100

Compilation du TP:

   Nombre d'erreurs: 0
   Nombre d'avertissements: 1

Tests JUnit du TP:

   Nombre d'erreurs: 4
   Nombre d'echecs: 0
   Nombre de tests effectues: 20
   Nombre total de tests: 20

Note sur 100: 80

Detection de plagiat:

   Nombre de cas suspects: 1
   Nombre total de cas verifies: 12


-----------------------------------------------------
DETAILS D'EXECUTION
-----------------------------------------------------

Compilation du TP:

   <Output typique de javac ici>

-----------------------------------------------------

Tests JUnit du TP:

   <Output typique de JUnit ici>

-----------------------------------------------------

Detection de plagiat:

   Nombre d'equipes a verifier: 12
   ...
   Cas suspects:
      AAAA12345678:BBBB12345678/tp3.txt

   Nombre de cas suspects: 1/12
```

3.2 Rapport diagnostique

Pour mieux comprendre ce qui ne va pas dans le fonctionnement d'un script, on peut demander un rapport diagnostique. Ce mode ignore le caractère privé/public et considère alors que tout est public, donc affiche tous les éléments du scripts (variables, tâches).

3.3 Rapport de groupe

Le rapport de groupe regroupe en fait l'ensemble des rapports individuels pour chacune des équipes. Cet ensemble de rapports est précédé par un compte-rendu de groupe. Ce compte-rendu liste l'ensemble des équipes et le résultat pour chacune d'elle, tel qu'il apparaît dans le rapport individuel. La liste des étudiants du groupe qui n'ont pas remis de TP est aussi fournie. Ce compte-rendu débute par une en-tête. Les groupes sont listés par ordre alphanumérique de leur identifiant d'équipe. Les membres d'un groupe sont listés par ordre alphanumérique de leur identifiant (code permanent). Les rapports apparaissent dans le même ordre.

Note: Il n'y a pas de mécanisme de pagination prévu bien que ce serait souhaitable dans une version future.

4 Gestion des erreurs

4.1 Causes principales d'erreur

Un script abstrait (donc syntaxiquement correct) arrête son exécution dès que l'une des conditions suivantes se produit:

1. Une tâche ne peut pas être menée à bien.
2. Le composant associée à une tâche n'existe pas.
3. Une référence fait appel à une variable ou à un attribut inexistant.
4. Une expression ne peut être évaluée (elle comporte une erreur).
5. Un fichier n'existe pas lors de la vérification de l'existence du fichier.
6. Une assertion s'avère fausse.

4.2 Causes d'échec pour les tâches

Les principales causes d'erreurs dans l'exécution d'une tâche sont :

a. La tâche tente d'accéder à une ressource à laquelle elle n'a pas accès (e.g., un fichier).

b. La tâche a épuisé ses ressources de stockage sur disque, son quota.

c. La tâche ne se termine pas dans les délais alloués (e.g., elle est prise dans une boucle infinie).

À ces erreurs externes s'ajoute des erreurs de programmation du composant :

d. Erreur interne.

e. Erreur d'interface avec Oto.

Les mécanismes utilisés pour détecter ces erreurs sont les suivants :

- Le composant lève une exception.
- Le minuteur pour tout le script a expiré.

4.3 Sortie en cas d'échec

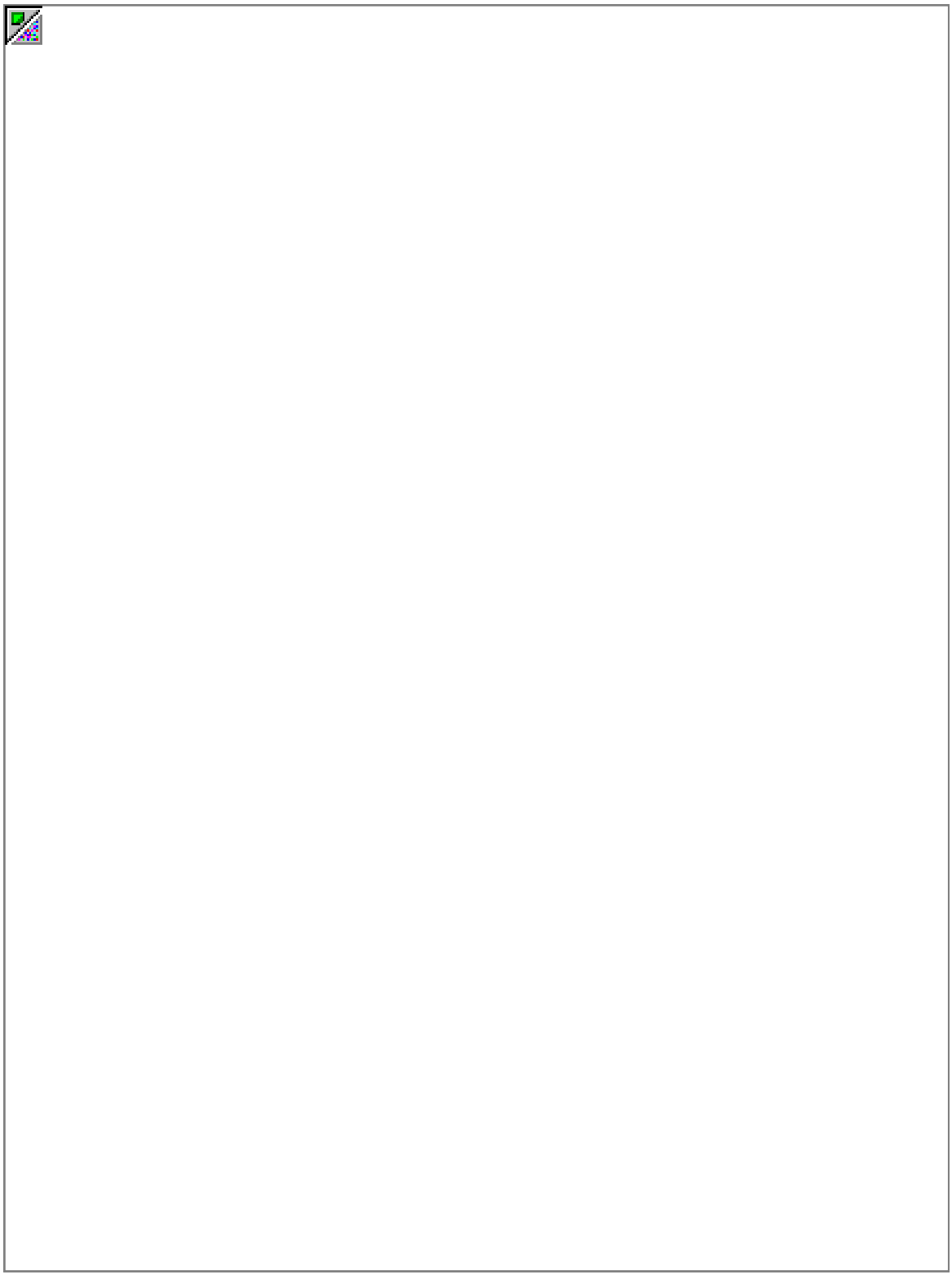
Le rapport d'exécution du script est produit pour toutes les étapes qui ont été exécutées convenablement. La cause d'erreur est fournie comme résultat sommaire du script.

Si l'erreur provient d'Oto lui-même (exception interne), le rapport n'est pas produit.

5 Syntaxe concrète

```
scriptOto ::= (declaration | assertion)* {sortie}

declaration ::= {description} {visibilite} declarationVarOuTache

assertion ::= 'assurer' expression 'sinon' expression

sortie ::= 'sortir' '{' ID (',' ID)+ '}'

description ::= '<<' CHAINE_ASREF_DESCRIPTIVE '>>'

visibilite ::= { '+' | '-' }

declarationVarOuTache ::= variable | variableFichier | tache

variable ::= ID '=' expression

variableFichier ::= ID '=?' expression

tache ::= ID '::' NOM_COMPOSANT '{' parametres '}'

parametres ::= variable*

expression ::=
   CHAINE_ASREF_IMPLICITE |
   expressionAinterpreter |
   chaineExplicite

expressionAinterpreter ::= '$(' CHAINE_ASREF_A_INTERPRETER ')'

chaineExplicite ::= '"' CHAINE_ASREF_EXPLICITE '"'

identificateurAsubstituer ::=
   '$' ID { '.' ID } |
   '${' ID { '.' ID } '}'

ID ::= [a-zA-Z_][a-zA-Z0-9\-_]*
NOM_COMPOSANT ::= ID ( '/' ID )*
CHAINE_ASREF_DESCRIPTIVE ::= tout sauf '>>' (mais avec '\>')
CHAINE_ASREF_IMPLICITE ::= tout jusqu'au prochain blanc
CHAINE_ASREF_A_INTERPRETER ::= tout, y compris '(...)' (ou '\)')
CHAINE_ASREF_EXPLICITE ::= tout sauf '"' (mais avec '\"')
```

6 Structure abstraite d'un script



7 Syntaxe abstraite

7.1 Exemple

```
variable : 1 : public : "Nombre total de points" : total_sur : "100"

fichier : 5 : prive : "Le fichier source fourni par l\'equipe" : fmain
: "tp3\.java"

fichier : 9 : prive : "Le fichier test fourni par le prof" : ftest :
oto . repscript + "\/tests\/testbase\.class"

tache : 13 : public : "Compilation du TP" : compilation : java/compile
: mainfile = fmain

assertion : 20 : ruby % compilation . nb_erreurs + " \=\= 0" : "La
compilation a echoue."

tache : 26 : public : "Tests JUnit du TP" : analyse : java/test :
fcible = compilation . mainclass , ftest = ftest

variable : 30 : public : "Note sur " + total_sur : note : ruby %
total_sur + " \- " + total_sur + " \* " + analyse . nb_erreurs + " \/
" + analyse . nb_tests_total

...

sortie : 50 : note , nb_cas_suspects
```

7.2 Grammaire

```
programme ::= ( { instruction | COMMENTAIRE } '\n' )*

instruction ::= tache | variable | fichier | assertion | sortie

tache ::=
   'tache' AVEC
   no_ligne AVEC
   publicOuPrive AVEC
   descripteur AVEC
   nom AVEC
   composant AVEC
   parametres

variable ::=
   'variable' AVEC
   no_ligne AVEC
   publicOuPrive AVEC
```

```
   descripteur AVEC
   nom AVEC
   expression

fichier ::=
   'fichier' AVEC
   no_ligne AVEC
   publicOuPrive AVEC
   descripteur AVEC
   nom AVEC
   expression

assertion ::=
   'assertion' AVEC
   no_ligne AVEC
   condition AVEC
   message

sortie ::=
   'sortie' AVEC
   no_ligne AVEC
   resultats

no_ligne ::= NOMBRE_ENTIER_POSITIF

publicOuPrive ::= 'public' | 'prive'

descripteur ::= chainesEtReferences

nom ::= ID_VARIABLE

composant ::= ID_COMPOSANT

condition ::= expression

message ::= expression

resultats ::= { nom ( ET nom )* }

parametres ::= { simpleParametre ( ET simpleParametre )* }

simpleParametre ::= nom EGAL expression

expression ::= { 'ruby' APPLIQUE_A } chainesEtReferences

chainesEtReferences ::= chaineOuReference (CONCAT chaineOuReference)*

chaineOuReference ::= CHAINE | reference

reference ::= nom { POINT  nom }

ET ::= ' , '
```

```
CONCAT ::= ' + '
EGAL ::= ' = '
AVEC ::= ' : '
APPLIQUE_A ::= ' % '
POINT ::= ' . '

COMMENTAIRE ::= ESPACE* '#' TOUT_CARACTERE_SAUF_FDL*

NOMBRE_ENTIER_POSITIF ::= [1-9][0-9]*
ID_VARIABLE ::= [a-zA-Z_][a-zA-Z0-9\-_]*
ID_COMPOSANT ::= ID_VARIABLE ( '/' ID_VARIABLE )*
CHAINE ::= '"' ( ALPHANUM | ESPACE | ECHAP_SYMBOLE)* '"'
ECHAP_SYMBOLE ::= '\' SYMBOLE
SYMBOLE ::= IMPRIMABLE - (ALPHANUM | ESPACE)
ALPHANUM ::= [a-zA-Z0-9]
ESPACE ::= [ \t]
```